Integrating open-ended learning in the sense-plan-act robot control paradigm

ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200373

Integrating Open-Ended Learning in the Sense-Plan-Act Robot Control Paradigm[1]

Angelo Oddi and **Riccardo Rasconi** and **Vieri Giuliano Santucci** and **Gabriele Sartor** and **Emilio Cartoni** and **Francesco Mannella** and **Gianluca Baldassarre** [2]

Abstract. This paper presents the achievements obtained from a study performed within the IMPACT (*Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs*) Project funded by the European Space Agency (ESA). The main contribution of the work is the realization of an innovative robotic architecture in which the well-known *three-layered* architectural paradigm (decisional, executive, and functional) for controlling robotic systems is enhanced with *autonomous learning* capabilities. The architecture is the outcome of the application of an interdisciplinary approach integrating Artificial Intelligence (AI), Autonomous Robotics, and Machine Learning (ML) techniques. In particular, state-of-the-art AI *planning systems* and algorithms were integrated with Reinforcement Learning (RL) algorithms guided by *intrinsic motivations* (curiosity, exploration, novelty, and surprise). The aim of this integration was to: (i) develop a software system that allows a robotic platform to autonomously represent in symbolic form the *skills* autonomously learned through intrinsic motivations; (ii) show that the symbolic representation can be profitably used for automated planning purposes, thus improving the robot's exploration and knowledge acquisition capabilities. The proposed solution is validated in a test scenario inspired by a typical space exploration mission involving a rover.

1 Introduction

The IMPACT (*Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs*) project aims at investigating the possibility of employing Artificial Intelligence (AI) techniques to increase both the cognitive and operational *autonomy* of artificial agents in general, and of robotic platforms targeted at the space domain in particular. The idea is based on the creation of a virtuous loop in which the agent increases its learned competence and knowledge through a direct interaction with the real environment, and then exploits the autonomously acquired knowledge to execute activities of increasing complexity. This process is cumulative and virtually *open-ended* [1, 5] since the information and abilities acquired up to a certain time are employed to further increase the agent's knowledge on the application domain, as well as the skills to adequately operate in it. This self-induced tendency towards autonomously learning new skills, based on *intrinsic motivations* (IM) [1, 7, 19], will enable the agent to face situations and solve problems not foreseeable when the agent is designed and implemented, especially because of the limited knowledge on the environment the agent will operate in.

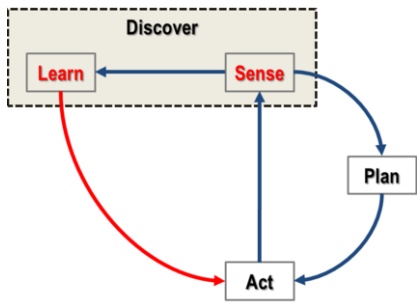

Figure 1: The *Discover-Plan-Act* cycle, extending the known SPA paradigm by adding the *Learn* capability between the *Sense* and the *Act* step.

The IMPACT idea proposes a possible extension of the well-known robot control architecture [2, 6, 9] formed by *Decisional*, *Executive* and *Functional* layers, commonly adopted within autonomous robotics to support the *Sense-Plan-Act* (SPA) autonomous deliberation and execution cycle. The architecture we propose realizes a *Discover-Plan-Act* (DPA) cycle (see Figure 1) which directly extends the SPA cycle with a more general open-ended learning step (*Discover*) directed to acquire new knowledge from the external environment. In particular, the innovative aspect of our contribution is the addition of an autonomous learning capability in the three-layered architecture, implementing the following features:

1. Autonomous learning of new *skills* based on self-generated goals driven by intrinsic motivations (intrinsic goals) [4, 22];
2. Automatic *abstraction* of the newly acquired skills [11, 16], from a low-level (sub-symbolic) to a high-level symbolic representation, e.g. expressed in *Planning Domain Definition Language – PDDL* [14]);
3. Autonomous enrichment of the planning domain by adding knowledge on new states and operators expressed through the high-level generated symbols [15].

This DPA architecture was integrated with a simulated robot platform implemented using the Gazebo Robot Simulator [10] [3] used to test the architecture. The test in particular involved a robot arm scenario inspired by a typical space exploration mission involving

[1] This research has been supported by the European Space Agency (ESA) under contract No. 4000124068/18/NL/CRS, project IMPACT - Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs - and the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 713010, Project "GOAL-Robots – Goal-based Open-ended Autonomous Learning Robots". The view expressed in this paper can in no way be taken to reflect the official opinion of the European Space Agency.
[2] ISTC-CNR, Via San Martino della Battaglia 44, 00185 Rome, Italy. Email: {angelo.oddi, riccardo.rasconi, vieri.santucci, gabriele.sartor, emilio.cartoni, francesco.mannella, gianluca.baldassarre}@istc.cnr.it
[3] http://gazebosim.org/

a rover. The scenario demonstrates the ability of the architecture to discover new ways to interact with the environment, abstract a high-level representation of the newly acquired capabilities, and integrate them in its planning domain.

The rest of the paper is organised as follows. Section 2 provides a top level description of the IMPACT framework concepts. Section 3 evaluates its performances in the test scenario. Finally, Section 4 provides some conclusions and discusses some possible directions of future work.

2 The IMPACT Architecture

Figure 2 shows the high-level software architecture of the IMPACT framework. The framework is based on the *three-layer* robot control architecture [2, 6, 9] commonly accepted in autonomous robotics to support the *Sense-Plan-Act* (SPA) autonomous deliberation and execution paradigm. In this architecture, the *decisional* layer implements the high-level planning, plan execution, and (re)planning capabilities; the *executive* layer controls and coordinates the execution of the functions distributed in the software, according to the requirements of the high-level tasks that have to be executed; lastly, the *functional* layer implements all the basic, built-in robot sensing, perception, and actuation capabilities. With respect to the classical structure of the architecture, Figure 2 presents the new blocks representing the proposed extensions that implement the learning capabilities, highlighted in red. In the following subsections, each layer of the architecture containing these blocks will be described in detail.

2.1 The Decisional Layer

Within the *classical* framework of the three-layered architecture, the decisional layer contains a task planner that generates a sequence of operations whose execution reaches a *goal* provided by the users. The IMPACT project tackled the main challenge of integrating this planning capabilities with autonomous, intrinsically motivated learning (IM-learning) algorithms for increasing the number of the robot's skills, and the automatic extension of the related high-level knowledge base. It is worth noting that this novel vision of the three-layered architecture can be seen as one of the possible ways for addressing the *long-term autonomy* (LTA) problem for robotic systems [13]. In general, LTA can be seen as: (i) the ability of a robotic system to perform reliable operations for long periods of time under changing and unpredictable environmental conditions; (ii) the capability of autonomously increasing knowledge about the working environment. Within the previous LTA interpretation, we considered the integration of IM-based learning capabilities as one of the possible "enablers" for long-term robot autonomy.

As Figure 2 shows, the decisional layer is composed of mainly two modules: (i) the ROSplan system (proposed in [3]), and (ii) the novel long-term autonomy manager (LTA-M).

ROSPlan ROSPlan is composed of two main ROS [17] nodes: (1) the Knowledge Base (KB) and the (2) Planning System (PS), and implements the above mentioned *Sense-Plan-Act* (SPA) autonomous deliberation and execution paradigm (such cycle is named *Knowledge-gathering*, *Planning*, and *Dispatch* in [3], but it substantially coincides with SPA) that traditionally characterizes autonomous robotics. In particular, the PS realizes both the automated planning and dispatching capabilities, while the KB is a collection of interfaces intended to store the up-to-date high-level model of the environment. *Planning* is the well-known process of generating a sequence of actions a (as instances of PDDL operators, to be described later) to achieve a given goal from an initial state, whereas *dispatching* controls and coordinates the execution of the functions distributed over the various functional level modules according to the task requirements. To summarize, the PS: (i) synthesizes the high-level PDDL representation of the initial low-level state leveraging the data stored in the KB, thus allowing to create a PDDL problem instance; (ii) passes the PDDL problem instance to the planner; (iii) dispatches each action, and decides when to reformulate and *re-plan*.

ROSPlan realizes the dispatching process by selecting the activities belonging to the current plan one-by-one, and passing them to the *Executive layer* module, which basically mediates between the decisional and the functional layer, activating or deactivating the reactive functions according to the planner's specification.

In the typical SPA cycle, the execution process can either terminate with a success, in which case the dispatching passes to the next activity if one exists, or a failure, in which case a re-planning process is triggered by the LTA-M based on a reformulation of the problem. As opposed to this typical SPA cycle, when an execution failure is returned our DPA architecture allows the LTA-M to possibly command a new *learning process*, in addition (or prior) to commanding the replanning. An example of learning process triggered by the LTA-M will be presented in Section 3, in relation to the robotic-arm scenario.

LTA-M The LTA-M represents the architecture component that provides a set of strategies to coordinate the achievement of both *extrinsic* and *intrinsic goals*, i.e. respectively the mission goals coming from Mission Control and the learning goals generated by curiosity driven behaviours respectively, thus realizing the overall system's long-term autonomy behavior. In short, the LTA-M is the component that decides when the system will operate in the ordinary SPA execution mode, exchanging information with the ROSPlan module, and when it will operate in the *Discover-Plan-Act* (DPA) execution mode, exchanging information with the GRAIL-IM learning module.

As we will see, the DPA cycle of the IMPACT architecture represents a significant extension of the SPA cycle, introducing an open-ended learning step (*Discover*) that basically implements the process of autonomous enrichment of the planning domain by adding knowledge on new states and operators expressed in high-level symbols.

2.2 The Functional Layer

The functional layer (called *skill layer* in [2]) has access to the system sensors and actuators and provides reactive behaviour which is robust even under environmental disturbances, for example with the help of closed-loop control. In the IMPACT architecture the functional layer has a twofold role (Figure 2): (1) it provides a set of stable controllers to operate in the environment (GRAIL-C component), and (2) it guides the autonomous learning of new skills on the basis of self-generated goals (GRAIL-IM-Learning component).

These two functions are guaranteed to implement a version of the GRAIL system [22], modified to take into consideration contextual features [20]. GRAIL leverages IMs to autonomously discover interesting states to be stored as possible goals. Moreover, it uses competence-based intrinsic motivations (CB-IMs; [21]) to select them. Goals are then used to drive the autonomous learning of the skills to be achieved. These skills are stored in data structures called *experts*. GRAIL is in itself a hierarchical system with different components: (i) a *goal-discovery* layer that recognizes relevant (new or unexpected) changes in the environment and stores the resulting states as possible goals; (ii) a *goal selector* that selects a

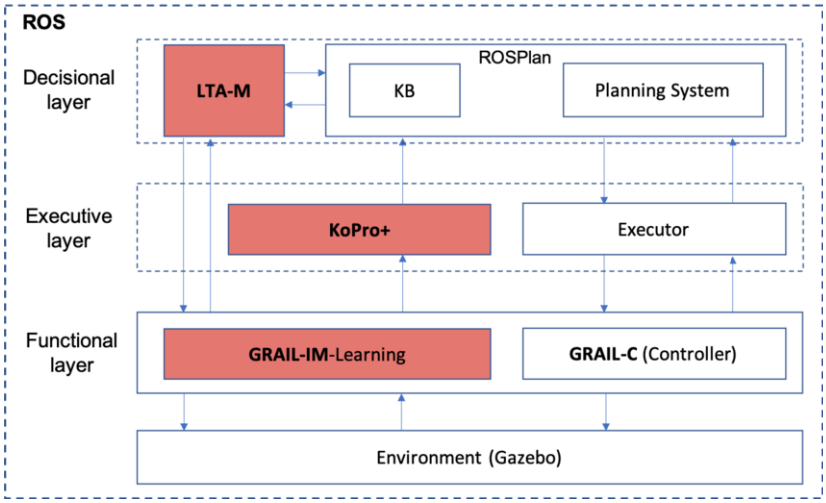

Figure 2: IMPACT high-level software architecture

goal, the skill related to which is trained, on the basis of the context and of CB-IMs (CB-IMs lead to selecting the goal whose skill has the highest competence improvement rate); (iii) a *goal-matching* component, that autonomously recognises the achievement of the selected goal; (iv) the context-dependent *experts*, each storing the skill competence associated to a different goal; (v) a *predictor*, used to both assess the current skill competence on a specific goal (given a context) and to compute the CB-IMs signal used for goal-skill selection. These components constitute the GRAIL-IM-Learning unit in the functional layer, while GRAIL-C stores the acquired skills (the experts) resulting from the learning process.

GRAIL can be considered as a general architecture, a sort of blueprint, to perform autonomous open-ended learning. The mechanisms and functions described above can be implemented in different ways (e.g., [22, 24]) and their in-depth analysis is beyond the scope of this work. However, in sec. 3.4 we provide some details on how the specific GRAIL functions have been implemented within the IMPACT project to tackle the experimental scenario.

2.3 The Executive Layer

The executive layer mediates between the decisional and functional layer, i.e. it activates or deactivates the reactive functions according to the deliberator's directives. It controls and coordinates the execution of such functions distributed over the various functional level modules, i.e. the *experts*, according to the task requirements. The problem of mapping symbolic abstract plans to continuous actions in real working domains is well recognised in the literature [2, 6, 9].

As shown in Figure 2, in our architecture the executive layer has been enriched with the "KoPro+" subsystem[4], that has the task of automatically translating the newly acquired skills from a low-level sub-symbolic representation generated by the GRAIL-IM module, to a high-level symbolic representation. Indeed, the agent senses the environment through its sensors, which return low-level information only, generally represented by all the values of the continuous variables that constitute the low-level state representation. Consequently, the representation of the skills that the agent learns by direct experience with the environment will be initially represented in sub-symbolic terms, a representation not suitable for high-level reasoning. In particular, the learned skills are expressed at low-level by leveraging an *option-based* representation [26]. Each skill $o = \langle Cl(I), Cl(E) \rangle$ is characterized by two *classifiers* (e.g., implemented as *decision trees* [18]), namely the *Initiation Set* classifier $Cl(I)$ and the *Effect Set* classifier $Cl(E)$. In particular, given a skill o and a low-level state s, the $Cl(I)$ classifier is used to test whether o can start from s or not (i.e., s belongs to o's *initiation set*). Similarly, the $Cl(E)$ classifier is be used to test whether s is a possible "effect state" after executing o.

The classifiers for each learned skill o are trained on the basis of the data obtained by repeatedly executing o, and saving these data as `true` (positive case) or `false` (negative case) training instances, depending on the classifier's purposes. In particular, to build the $Cl(I)$ training dataset, we considered as positive cases all the low-level variable values before the successful execution of the skill, and as negative cases all the low-level variable values in conditions where GRAIL tried to execute the skill but the *predictor* (see Section 2.2) was always zero. To build the $Cl(E)$ training dataset, we considered as positive cases all the low-level variable values following a successful execution of the skill, while as negative effect cases we used all the low-level variable values before the execution of that skill, whether it succeeded or not (since we know that GRAIL will not execute the skill if its goal/effect is already achieved).

To summarize, the "KoPro+" procedure accepts in input a $\langle Cl(I), Cl(E) \rangle$ pair for each skill, and returns the complete set-theoretic PDDL representation (described below) related to the agent's currently learned low-level skill set. Every *proposition* of the returned PDDL domain corresponds to a *symbol* automatically produced by "KoPro+" out of the effects of each skill; all the produced symbols are saved in the set P. As "KoPro+" proceeds, a different classifier $Cl(\sigma_i)$ is created for each produced symbol $\sigma_i \in P$. Ultimately, the $Cl(\sigma_i)$ classifier will be used to determine σ_i's *truth value* every time it is necessary. Every symbol σ_i is associated to those subset of low-level states that determines σ_i's semantics (σ_i's *Grounding Set*).

As anticipated above, each skill learned by the agent will be abstracted into its related PDDL operator expressed in a Set-theoretic representation, whose preconditions and effects sets are subsets of the P set. Formally, a set-theoretic PDDL domain specification is expressed in terms of a set of propositional symbols $P = \{\sigma_1, ..., \sigma_n\}$ (each associated to a grounding classifier $Cl(\sigma_i)$) and a set of op-

[4] A thorough description of the "KoPro+" abstraction algorithm is beyond the scope of this paper; the interested reader may refer to [12].

erators $A = \{opt_1, ..., opt_m\}$. Each operator opt_i is described by the tuple $opt_i = \langle pre_i, eff_i^+, eff_i^- \rangle$ with $pre_i, eff_i^+, eff_i^- \subseteq P$, where pre_i contains all the propositional symbols that must be $true$ in a state s to allow the execution of opt_i from s, while eff_i^+ and eff_i^- contain the propositional symbols that are respectively set to $true$ or $false$ after opt_i's execution. All the other propositional symbols remain unaffected by the execution of the operator.

For example, let us consider the following operator, called `graspObject`, whose action corresponds to grasping a stone-shaped object on behalf of a robotic arm (more information about the related PDDL domain will be given in Section 3):

```
(:action opt_4
    :parameters ()
    :precondition (and (Symbol_0) (Symbol_7)
    (Symbol_9))
    :effect (and (Symbol_5) (not (Symbol_7)))
)
```

As immediately observable, the operator follows the standard set-theoretic PDDL, where $pre_i = \{Symbol_0, Symbol_7, Symbol_9\}$, $eff_i^+ = \{Symbol_5\}$, and $eff_i^- = \{Symbol_7\}$ (note that the name of the symbols is automatically generated). In order to provide the reader with some information about the meaning of the symbols that populate the previous operator, the semantics of `Symbol_0` proposition in the operator's preconditions is "*object in sight*" (the object must be in sight of the agent in order for the agent to grasp it). More specifically, `Symbol_0`'s grounding classifier will return the $true$ value only from the low-level states sensed by the robot as having the v_3 variable in the $true$ range (see Figure 5: the v_3 pixel returns the image of an object within the rover's grasping distance). The remaining symbols have the following semantics (the related explanations are omitted for reasons of space): `Symbol_7` = "*object not grasped*"; `Symbol_9` = "*object not stowed*"; `Symbol_5` = "*object grasped*".

We lastly observe that the "KoPro+" module has a *dual role* in comparison to the Executor; in fact, the learning process can be seen as opposite to the execution process as it maps low-level sub-symbolic representation into abstract symbolic representation.

3 Operational Evaluation

This section is dedicated to describing the behavior of our architecture within a simulated test case, called *Robotic Arm* scenario. The *Robotic Arm* test case entails the presence of an exploration rover equipped with a gripper actuator attached to a manoeuvrable arm, trying to grasp a "vase shaped" rock whose size exceeds the max opening span of the gripper. The robot is thus not able to pick-up the rock using its basic grasping skill; however, upon a "surprising" failure (i.e., a failure occurring when performing a skill supposed to be successful), the system will automatically trigger the learning of a new skill and at the end of the learning process the robot will be able to pick-up the "vase shaped" rock. Finally, the new skill will be abstracted in terms of a new PDDL operator, and will be seamlessly added to the existing PDDL domain, ready to be used by the planner.

In this scenario, the rover is supposed to explore a region of the Mars surface divided in two zones ($zone_1$ and $zone_2$); both zones can be reached through a single waypoint ($zone_3$, see Figure 3). The Gazebo model of the robotic arm used for this test case is a simulation of a KUKA robotic arm (called Youbot [5]). The behaviour

[5] https://github.com/islers/youbot_simulation

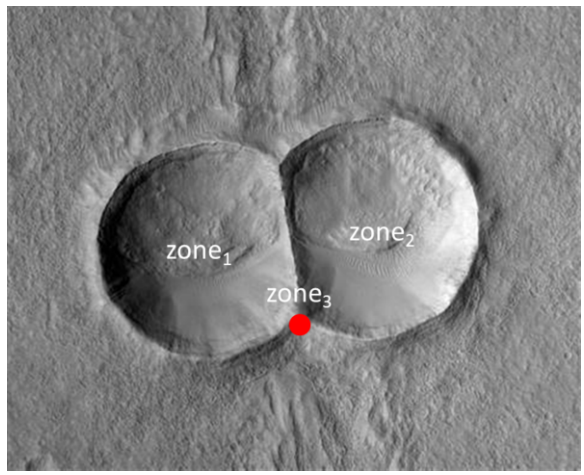

Figure 3: The scenario selected for the Robotic Arm use case, representing an area of the martian terrain characterized by two twin craters, that the rover should explore for sample grasping purposes. It is possible to travel from $zone_1$ to $zone_2$, and vice versa, through $zone_3$ (depicted in red).

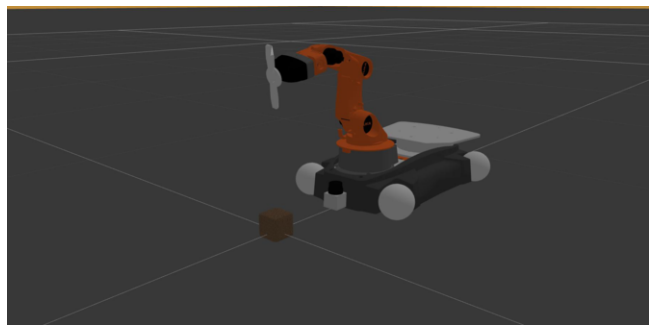

Figure 4: The Rover Model used in the Robotic Arm Scenario.

of the simulator is basically intended to mirror the behaviour of the real Youbot (see Figure 4), including the physics and all the topics and exchanged messages. The Youbot arm is used to perform all the grasping activities, as well as the stow activities. In particular, the Youbot model will be used during the learning process of the new grasping activity.

3.1 Low-level Operators and variables description

At the beginning of the test case, the rover is supposed to be equipped with a pre-compiled PDDL planning domain (described in Section 3.2), composed of the following high-level operators:

- `moveBeforeObject_1` (`opt_0`): the rover moves from $zone_3$ to $zone_1$ and places itself in front of an object, in a position suitable for grasping it;
- `moveBeforeObject_2` (`opt_1`): the rover moves from $zone_3$ to $zone_2$ and places itself in front of an object, in a position suitable for grasping it;
- `moveBackFrom_1` (`opt_2`): the rover moves from $zone_1$ to $zone_3$;
- `moveBackFrom_2` (`opt_3`): the rover moves from $zone_2$ to $zone_3$;
- `graspObject` (`opt_4`): the rover activates the robotic arm to grasp the object that is in front of it;

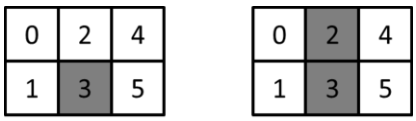

Figure 5: The view field of the rover is composed of 6 low-level variables (pixels) numbered from 0 to 5. The dark pixels represent that an object is in sight; the number of darkened pixels will provide some simplified information about the type of the observed object: stone-shaped object (left), vase-shaped object (right).

- `stowObject` (`opt_5`): the rover uses the robotic arm to stow a grasped object inside the rover's onboard container;
- `readyGripper` (`opt_6`): the rover resets its robotic arm in a position suitable for the next grasping (the gripper is supposed to hold no object).

We will use this scenario to provide an example of autonomously synthesized "enrichment" of the current PDDL domain representation, as a consequence of an intrinsically motivated learning activity through a direct interaction (i.e., sensing + acting) with the environment. The final result will be a version of the PDDL planning domain automatically extended with a *new* grasping operator that allows the rover to grasp differently shaped objects in a different fashion w.r.t. the original grasping.

The complete low-level variables set S is composed of 10 variables $S = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ under continuous observation on behalf of the rover, whose meaning is the following:

- $v_0, ..., v_5$: low-level variables that constitute the rover's view field. Each such variable corresponds to a single pixel, and therefore the view field will be composed of a 2×3 pixel matrix (see Figure 5)[6];
- v_6: low-level variable that senses the grasping of an object ($v_6 = false \rightarrow$ no object is grasped, $v_6 = true \rightarrow$ an object has been grasped and it is being held by the gripper);
- v_7: low-level variable that senses the presence of an object in the stowing bucket ($v_7 = false \rightarrow$ no object is stowed, $v_7 = true \rightarrow$ the object is being stowed);
- v_8: low-level variable that senses the rover's being $zone_1$ ($v_8 = false \rightarrow$ the rover is not in $zone_1$, $v_8 = true \rightarrow$ the rover is in $zone_1$);
- v_9: low-level variable that senses the rover's being in $zone_2$ ($v_9 = false \rightarrow$ the rover is not in $zone_2$, $v_9 = true \rightarrow$ the rover is in $zone_2$);

In general, each low-level variable can take an infinite set of real values; without loss of conceptual generality w.r.t. KoPro+'s abstraction capabilities, we will however assume that such values will be translated to boolean values depending on the values captured by the rover's sensors. For instance, if the position tracking system (returning the x and y coordinates of the rover) detects that the rover is *sufficiently* close to the center of $zone_1$, then the variable v_8 (which can be interpreted as a "*proximity to* $zone_1$" signal) will be set within the $true$ range, while the $zone_2$ proximity variable will be set within the $false$ range, and vice versa.

3.2 The Initial PDDL domain

The initial PDDL domain proposed for this scenario is supposed to be automatically synthesized by means of the "KoPro+" abstraction procedure. In the following, we provide an example of utilization of such PDDL domain, presenting a solution of a PDDL problem representing a typical mission goal that the rover is supposed to fulfill once on the martian surface. The problem, expressed in PDDL syntax, is the following:

```
(define (problem marsprob) (:domain RobGripper)
(:init (Symbol_1) (Symbol_4)
       (Symbol_6) (Symbol_7) (Symbol_9))
(:goal (and (Symbol_8) (Symbol_2)))
)
```

This problem's initial condition entails the rover starting from $zone_1$ (`Symbol_1` = "*rover in* $zone_1$" and `Symbol_4` = "*rover not in* $zone_2$"), with the object not yet visible (`Symbol_6` = "*object not in sight*"), nor grasped (`Symbol_7` = "*object not grasped*") or stowed (`Symbol_9` = "*object not stowed*"). The problem's goal is stowing an object (`Symbol_8` = "*object is stowed*") in $zone_2$ (`Symbol_2` = "*rover in* $zone_2$").

The solution, obtained by submitting the problem to an off-the-shelf planner, is the following:

```
SOLUTION PLAN:
opt_2:   [moveBackFrom_1]
opt_1:   [moveBeforeObject_2]
opt_4:   [graspObject]
opt_5:   [stowObject]
```

The rover moves from $zone_1$ to $zone_3$ (`opt_2`) and then from $zone_3$ to $zone_2$ (`opt_1`) in front of the object, then grasps the object using its grasping skill (`opt_4`), then stows the object (`opt_5`).

3.3 Automatically synthesizing the new PDDL domain

Let us now suppose that, unknowingly to both the mission's managers and the rover's designers, the rover encounters a concave, vase-shaped object in $zone_2$ during the exploration. According to the rover's high-level logic, the object is still recognized as an interesting sample, and therefore the rover attempts to grasp it using the usual grasping skill. Let us now suppose that the object is wider than the maximum span of the gripper; *in this case, the grasping action will fail*. The failure is immediately captured by the functional layer of the architecture, and passed on to the LTA-M in the decisional layer. The LTA-M is therefore called to decide whether the failure requires a re-planning or a new learning process (see Section 2.1). If the LTA-M opts for the second case, the failure thus detected will trigger a new learning goal, e.g., driven by the surprise deriving from having failed the execution of a skill whose reliability was held high. The net result is that the rover will be autonomously prompted to initiate a process to learn how to grasp the new object, leveraging the GRAIL-IM module, as described in the next section. When the learning process is terminated, a new skill will be acquired where the rover is now capable to grasp the vase-shaped object (e.g., by pinching it by its edge, or by inserting the closed gripper in the vase and then opening it, depending on the particular evolution of the learned policy).

After the learning of the new skill and before the synthesis of the new PDDL domain, a period of environment exploration or "execution rehearsal" of all the known skills (old and new) is commanded by the LTA-M, in order to acquire all the information to train the

[6] In general, a camera view field is composed by a much larger array of pixels; we use a small set without loss of generality within the scope of this work.

new classifiers necessary to build the new domain[7], as described in Section 2.3. When the LTA-M decides that the amount of training data is sufficiently high, the $Cl(I)$ and $Cl(E)$ classifiers' training is triggered anew for all the old skills as well as for the one newly acquired. As a last step, the "KoPro+" abstraction procedure can be commanded again by the LTA-M, and the new PDDL domain is autonomously synthesized. In the remainder of this section, only the differences between the old and the new PDDL domain will be highlighted, for reasons of space.

The first important difference between the two domain representations is the presence in the new domain of an extra symbol (i.e., `Symbol_10`) whose semantics is "Object is VASE". This symbol has been created as a consequence of the learning of the new grasping skill, and it is very important as it represents the symbolic element that discriminates between the two different classes of objects that the rover has encountered during its exploration, i.e., the *only* class of objects the rover was supposed and trained to operate on (the stone-shaped rocks), and the class of objects that the rover has incidentally discovered (the vase-shaped objects) and whose grasping had to be learned. The presence of this new symbol in the PDDL high-level representation of the domain is of great importance as it allows the planner to correctly synthesize plans that involve the grasping of either kinds of object *directly at symbolic level*. In other words, it allows the planner to produce plans that entail the utilization of the correct grasping operator, depending on the class the object to be grasped belongs to.

The second important difference is that in the new PDDL domain a new high-level operator is created: `graspObject_2`, synthesized as follows:

```
(:action opt_7
    :parameters ()
    :precondition (and (Symbol_0) (Symbol_10)
    (Symbol_7) (Symbol_9))
    :effect (and (Symbol_5) (not (Symbol_7)))
)
```

This operator is the high-level representation of the newly learned low-level skill, and therefore it is the operator that will be used to grasp the vase-shaped objects. Most interestingly, it can be readily confirmed that the newly created `Symbol_10` ("Object is VASE") has been added to the preconditions of the `graspObject_2` operator, thus addressing the fact that, in order for the new grasping operator to be activated, the rover not only has to find itself in front of an object (i.e., `Symbol_0` must be verified), but it must also be a vase-shaped object. The new PDDL domain will be further analyzed in Section 3.5.

3.4 Autonomous learning of new skills

Implementation of GRAIL in the experimental scenario. We now briefly describe the implementation of the mechanisms and functions introduced in Section 2.2, and that are actually involved in the current experimental scenario.

In the Robotic Arm scenario we assume that the system is already endowed with a series of different goals and with the related skills necessary to achieve them (see Section 3.1) from a generic starting condition (the *context*). Here, a goal is intended as a state (or an effect) the system could try to achieve. If a state s is represented as a set of n features $\{f_1, f_2..., f_n\}$, a goal can be a state where a certain feature (e.g. f_2) assume a certain value x. In this section we are focusing on the goal related to the grasping of an object positioned in front of the robot, and therefore the *goal discovery* and *goal selection* mechanisms can be considered "switched off", so that the system is continuously selecting the same goal. In general however, the autonomous motivational system is fully operational in the architecture, providing the goal with the intrinsic motivations that would drive the system in pursuing it, even when competing with other possible tasks. Different contexts are associated to each goal stored by the system, which is thus able to compare them and recognise if a new context is encountered. In this specific implementation, the context is a set of sensory inputs the system is perceiving: the two different visual inputs in Figure 5 might constitute two different contexts when selecting an expert to perform a skill. In the particular case we are tackling here, we simply assume that the system has one expert for each context; whenever a new situation is encountered, a new context-related "branch" is created together with a new expert that is a copy of a previous expert used for the same goal. The experts control the arm and the gripper of the robot and, given the selected goal and the current context, they learn the policy necessary to achieve it. Experts are currently implemented through discrete-movement Dynamic Movement Primitives (DMPs, [8, 23]), i.e. dynamical models that can generate a movement trajectory on the basis of end-position parameters (controlling arm and gripper final postures) and shape parameters (controlling the shape of the trajectory). In particular, each controller is composed of a set of 9 DMPs, each one used to control a different joint of the robotic arm and the activity of the gripper (9 DoF). Training trials are composed of 10 attempts. For each attempt, a Gaussian noise is generated and added to the parameters of the DMPs to generate new trajectories. The noise is dependent on expert performance: the higher the performance, the smaller the noise. After each attempt, DMPs are updated through Policy Improvement Black Box (PI^{BB}, [25]), that modifies the parameters of the DMPs

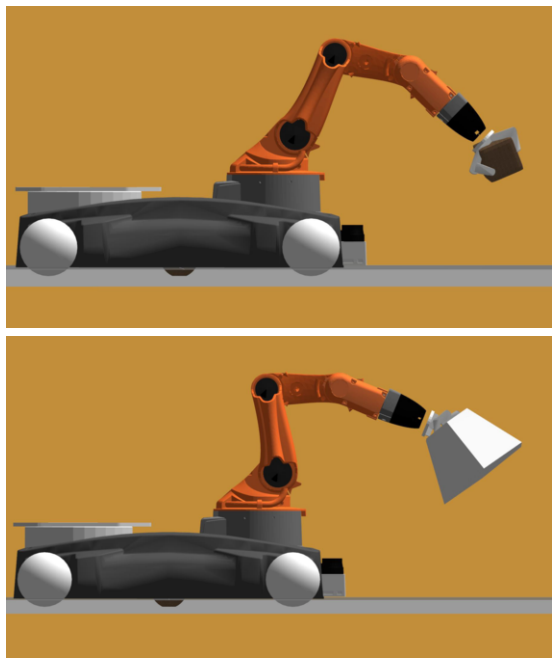

Figure 6: The rover, the Youbot arm and the gripper in the Robotic Arm scenario. [Up] The robot grasping the cube-shaped rock. [Bottom] The robot grasping the vase-shaped rock.

[7] In our system, the training information is stored in the usual Attribute Relationship File Format (ARFF).

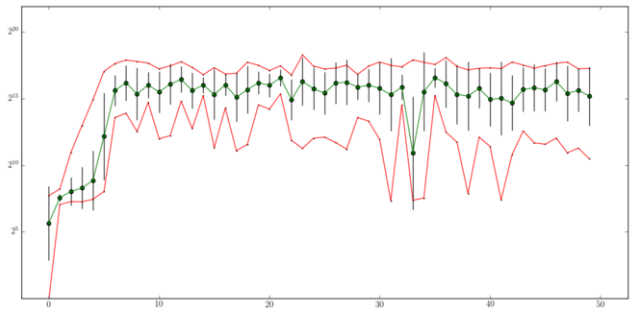

Figure 7: The learning curve of the new skill. Red curves indicate the performance of the best and the worse attempt (within a trial of 10 attempts). The green curve indicates the average performance within the trial, and whiskers show the standard deviation. Because the performance grows geometrically, the scale of the y axis is logarithmic.

composing the policy on the basis of a path-integral reward function for each attempt.

When an expert is used, the representation of the related goal is pre-activated so that GRAIL is able to check the achievement of the goal through a goal matching mechanism. As described in Section 2.1, when a failure occurs the LTA-M can allow GRAIL to interact with the environment and try to acquire new competences. In particular, if the robot tries to grasp a bigger object ("the vase", see Figure 6, bottom) with its pre-designed skill, it will not be able to accomplish the associated grasping goal. This will generate a sequence of different effects: (i) a failure signal will be generated through the goal-matching function; (ii) the context in which the system is working is checked and since having the vase in front of the robot is different from having the rock, a new context is generated for the "grasping goal", together with a new expert; (iii) a novelty based intrinsic motivation signal is generated for that goal so that the system, even when more goals are present at the same time, is biased to re-select the same goal and possibly start to acquire the new skill needed in the new context; (iv) the robot will start to attempt the grasping goal, and the more the system will be able to improve its ability in generating a goal matching from the new starting condition, the more a CB-IM signal will be computed to drive the system towards focusing on the same goal and properly acquire the new skill. In particular, the CB-IMs are calculated on the basis of the system improvement in achieving a goal: in this sense, CB-IMs can be considered as a derivative (over a certain time-window) of the system performance on a task. This formulation leads to the effect that CB-IMs are present only when they are needed: if the system is able to systematically generate the desired goal or, on the contrary, it is not able to improve its competence, the IMs will eventually "fade away" and the robot will start selecting other goals, or the control will pass again on the higher level component of architecture.

Learning the new skill. Figure 7 shows the performance of the robot in grasping the vase-shaped rock. After a first failure that triggered the exploration and learning processes, the system starts trying to achieve the goal from a new context (the vase-shaped rock never encountered before). Using a copy of the expert trained to work with the small rock, at the beginning the robot is not able to achieve the goal. However, after the first successes due to exploratory noise, the robot improves its competence in the first 5 trials. Nonetheless, at the beginning the noise variance over the parameters of the DMPs is high because the acquired competence is still growing: as a consequence, the robot continues to explore through the parameters and its competence improvement is growing slowly. However, after other 5 trials the system starts to get persistent improvements, resulting in higher level of rewards and better average performance over the attempts.

Note that even when the robot has reached a proper competence with the new skill (after 30 trials, see red dots in Figure 7) the performance has still a high variance even though the variance of parameters is low or zero. This is due to the fact that the simulation has an inner, independent noise, caused by a non-stationary gap in communication between the controller and the simulator. Although on the one hand this seems to impair a perfect learning of the skills, on the other hand it might be considered as a replication of the interaction between the controller and a real robot, where a little amount of noise in the communication can never be avoided.

3.5 Planning over the new Robotic Arm PDDL Domain

The correctness of the new autonomously synthesized domain is proven by demonstrating that the agent is now able to reach the goal of grasping the vase-shaped object in $zone_2$, again starting from $zone_1$ (it can be easily shown that the previous goal of grasping and stowing the stone-shaped object is still correctly planned with the new domain, but we omit the analysis for reasons of space). Obviously, the PDDL problem will be expressed exactly in the same terms as the one presented in Section 3.2, under the assumption that the vase-shaped objects can be found in $zone_2$ only.

The obtained solution is the following:

```
SOLUTION PLAN:
opt_2: [moveBackFrom_1]
opt_1: [moveBeforeObject_2]
opt_7: [graspObject_2]
opt_5: [stowObject]
```

Again, it can be observed that the plan is correct; the rover moves from $zone_1$ to $zone_3$ (`opt_2`), then moves from $zone_3$ to $zone_2$ in front of the object (`opt_1`), then grasps the object using the correct new operator (`opt_7`), and then the object is finally stowed (`opt_5`).

4 Discussion and Future Work

This paper presents an architectural design of an extended version of the well-known three-layered architecture that typically implements the behavior of a robotic system. The proposed architecture realizes a *Discover-Plan-Act* cycle (DPA), thus enhancing the well-known *Sense-Plan-Act* (SPA) paradigm with a more general open-ended learning step (Discover). New functionalities have been added to the classical three-layered architecture: (i) a goal-discovering and skill-learning architecture (GRAIL) connected to the symbolic abstraction procedure (KoPro+) that creates a processing pipeline from the continuous to the symbolic environment representation; (ii) a long-term autonomy manager (LTA-M), that strategically coordinates the achievement of both the external mission goals and the internal goals generated by curiosity driven behaviors. The complete functionalities of the system have been tested on a scenario inspired by a typical planetary rover mission.

Despite the core of the proposed architecture is the integration of an open-ended low-level learning process and a symbols abstraction process, we believe that this integration is more than the mere sum of its parts. Indeed, the composition of both previous features in a

dynamic environment fosters the capability of integrating newly discovered with previously abstracted knowledge, thus boosting the environment discovery capabilities, which is an invaluable added value to long term autonomy. Notwithstanding the relative simplicity of the experimental scenario, this paper presents for the first time a complete and working architecture that spans from low-level autonomous exploration, discovery, and skill learning to high-level automated reasoning, thus demonstrating the capability to autonomously learn new symbols and operators by direct and autonomous interaction with the environment.

The development of open-ended learning as an enabling factor for long-term autonomy of robotic systems opens many possible branches for future work. We are currently working on a probabilistic version of the "KoPro+" abstraction procedure, capable of capturing the stochastic nature of more general real-world phenomena. A second interesting direction of work is the definition of extended *long-term autonomy strategies*, i.e., the integration of symbolic planning and open-ended learning to increase the ability of one agent to autonomously acquire new symbolic knowledge based on previously acquired symbolic models (*bootstrap learning* [5]). As a possible extension, it is possible to define strategies to generate symbolic plans for collecting new intrinsic goals to achieve. Other strategies can be designed for planning to reach different sets of preconditions for learning the achievement of a given goal.

REFERENCES

[1] Gianluca Baldassarre and Marco Mirolli, *Intrinsically Motivated Learning in Natural and Artificial Systems*, Berlin: Springer and Verlag, 2013.

[2] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Mark G. Slack, 'Experiences with an architecture for intelligent, reactive agents', *Journal of Experimental & Theoretical Artificial Intelligence*, **9**(2-3), 237–256, (1997).

[3] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras, 'Rosplan: Planning in the robot operating system', in *ICAPS 2015, the 25th International Conference on Automated Planning and Scheduling*, (2015).

[4] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh, 'Intrinsically motivated reinforcement learning', in *Advances in neural information processing systems*, pp. 1281–1288, (2005).

[5] Stephane Doncieux, David Filliat, Natalia Diaz-Rodriguez, Timothy Hospedales, Richard Duro, Alexandre Coninx, Diederik M. Roijers, Benoit Girard, Nicolas Perrin, and Olivier Sigaud, 'Open-ended learning: A conceptual framework based on representational redescription', *Frontiers in Neurorobotics*, **12**, 59, (2018).

[6] Erann Gat, 'Three-layer architectures', in *Artificial Intelligence and Mobile Robots*, eds., David Kortenkamp, R. Peter Bonasso, and Robin Murphy, 195–210, MIT Press, Cambridge, MA, USA, (1998).

[7] Todd Hester and Peter Stone, 'Intrinsically motivated model learning for developing curious robots', *Artificial Intelligence*, **247**, 170 – 186, (2017). Special Issue on AI and Robotics.

[8] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, 'Dynamical movement primitives: Learning attractor models for motor behaviors', *Neural Computation*, **25**(2), 328–373, (Feb 2013).

[9] Félix Ingrand, Simon Lacroix, Solange Lemai-Chenevier, and Frederic Py, 'Decisional autonomy of planetary rovers', *Journal of Field Robotics*, **24**(7), 559–580, (2007).

[10] N. Koenig and A. Howard, 'Design and use paradigms for gazebo, an open-source multi-robot simulator', in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pp. 2149–2154 vol.3, (Sept 2004).

[11] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez, 'From skills to symbols: Learning symbolic representations for abstract high-level planning', *Journal of Artificial Intelligence Research*, **61**, 215–289, (2018).

[12] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez, 'From skills to symbols: Learning symbolic representations for abstract high-level planning', *Journal of Artificial Intelligence Research*, **61**, 215–289, (2018).

[13] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník, 'Artificial intelligence for long-term robot autonomy: A survey', *IEEE Robotics and Automation Letters*, **3**(4), 4023–4030, (Oct 2018).

[14] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins, 'PDDL - The Planning Domain Definition Language', Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, (1998).

[15] Dana Nau, Malik Ghallab, and Paolo Traverso, *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[16] Angelo Oddi, Riccardo Rasconi, Emilio Cartoni, Gabriele Sartor, Gianluca Baldassarre, and Vieri Giuliano Santucci. Learning high-level planning symbols from intrinsically motivated experience, 2019. arXiv:1907.08313.

[17] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng, 'Ros: an open-source robot operating system', in *ICRA Workshop on Open Source Software*, (2009).

[18] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[19] Richard M. Ryan and Edward L. Deci, 'Intrinsic and extrinsic motivations: Classic definitions and new directions', *Contemporary Educational Psychology*, **25**(1), 54 – 67, (2000).

[20] Vieri Giuliano Santucci, Gianluca Baldassarre, and Emilio Cartoni, 'Autonomous reinforcement learning of multiple interrelated tasks', in *Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob2019)*, pp. 221–227, (Aug 2019).

[21] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli, 'Which is the best intrinsic motivation signal for learning multiple skills?', *Frontiers in neurorobotics*, **7**, 22, (2013).

[22] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli, 'Grail: A goal-discovering robotic architecture for intrinsically-motivated learning', *IEEE Transactions on Cognitive and Developmental Systems*, **8**(3), 214–231, (2016).

[23] Stefan Schaal, *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, 261–280, Springer Tokyo, Tokyo, 2006.

[24] Kristsana Seepanomwan, Vieri Giuliano Santucci, and Gianluca Baldassarre, 'Intrinsically motivated discovered outcomes boost user's goals achievement in a humanoid robot', in *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 178–183, (2017).

[25] Freek Stulp, Gennaro Raiola, Antoine Hoarau, Serena Ivaldi, and Olivier Sigaud, 'Learning compact parameterized skills with a single regression', 417–422, (2013).

[26] Richard S. Sutton, Doina Precup, and Satinder Singh, 'Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning', *Artificial Intelligence*, **112**(1-2), 181–211, (August 1999).